Distributed Query Optimization Using Multi-attribute Semijoin Operations

by

Qiuling Fu

A Thesis
Submitted to the Faculty of Graduate Studies and Research through the School of Computer Science in Partial Fulfillment of the Requirements for the Degree of Master of Science at the University of Windsor

Windsor, Ontario, Canada
1996
The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author’s permission.

L’auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L’auteur conserve la propriété du droit d’auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.
Qiuling Fu 1996
© All Rights Reserved
Abstract

The efficiency of query processing strategies is critical for system performance in any distributed database management system. Many query processing strategies have been proposed to minimize either the response time or the total cost, or both. In this thesis, we introduce the concept of multi-attribute semijoin (MASJ) — a new database operation to reduce the communication cost (ignoring the local cost of total cost) in distributed query processing. The objective of this investigation is to find out whether this operation gives significant improvements in the communication cost to process distributed queries. We explore some useful properties of this operation and develop a heuristic to identify situations where this operation is useful. Based on these results, we propose a query processing strategy called the MJ Algorithm to minimize the communication cost for query optimization. The MJ Algorithm combines the AHY Algorithm (total time version) and the multi-attribute semijoin operation. Our aim is to find out whether the new algorithm can construct better reducers than the AHY Algorithm. Finally, we use simulation studies with a large number of queries and our experiments indicate that the performance of the MJ Algorithm is significantly better than that of the AHY Algorithm.
To my mother, Jinan Song

my father, Zhihang Fu

my husband, Wei Ye

my sister, Qiumiao Fu

and my son, Toddy Ye
Acknowledgments

This thesis work could not have been accomplished without the support of many people. I would like to express my appreciation to Dr. Subir Bandyopadhyay, whose tremendous support, guidance, stimulating ideas, and review of this thesis report were invaluable to the successful completion of this thesis. I would like to thank Dr. Joan Morrissey for her guidance and suggestions on my survey related to this thesis, and her insightful comments on this thesis. Thanks to Dr. Diane Kao for her comments on this thesis. Thanks to W. T. Bealor for providing me with the source code for the AHY Algorithm, and the query and database generation programs. Finally, I would like to thank my friends, Donato Ingratta for his comments on this thesis report and encouragement, and Sandeep Kamat for providing me with many references and his technical support.
# Table of Contents

Abstract ........................................................................................................ iv  
Acknowledgments .......................................................................................... vi  
List of Figures .................................................................................................. x  
List of Tables ................................................................................................... xi  

## 1 Introduction

1.1 The Thesis Statement and Topics to Be Investigated ...................... 3  
1.2 Organization of The Thesis .............................................................. 4  

## 2 Review of Literature

2.1 Distributed Database System .......................................................... 5  
2.2 Distributed Query Processing and Optimization .......................... 7  
2.3 Cost Measures .................................................................................... 8  
2.4 Query Graph ...................................................................................... 9  
2.5 Distributed Database Operations ................................................. 10  
2.6 Estimating The Effect of Database Operations ......................... 12  
2.7 Query Processing Strategies ......................................................... 16  

### 2.7.1 Semijoin Strategies ................................................................. 17  

#### 2.7.1.1 The SDD-1 Strategy ...................................................... 17  
#### 2.7.1.2 The AHY Strategy ...................................................... 19
2.7.2 Join Strategies ........................................... 22
  2.7.2.1 Distributed INGRES Strategy .............. 22
  2.7.2.2 R* Strategy ........................................ 24

3 MJ ALGORITHM USING MULTI-ATTRIBUTE SEMIJOIN (MASJ)
  OPERATION .................................................... 28
  3.1 Notion of a Multi-attribute Semijoin (MASJ) Operation .... 28
  3.2 Promising Subquery Graph and Schedule With MASJ
      Operation (MSCH) ........................................ 34
  3.3 The MJ Algorithm — Our Heuristic for Query Processing . 40
      3.3.1 The Steps of The MJ Algorithm ................. 40
      3.3.2 Example 3.4 ........................................ 41
  3.4 Details of The MJ Algorithm ............................. 46
      3.4.1 Details of Step 1 ................................. 47
      3.4.2 Details of Step 2 ................................. 47
      3.4.3 Details of Step 3 ................................. 49
      3.4.4 Details of Step 4 ................................. 49

4 EXPERIMENTS AND EVALUATION ............................. 51
  4.1 Objectives of Our Test ................................. 51
      4.1.1 Generating The Set of Queries ................. 52
      4.1.2 The Test Database ............................... 54
4.2 Results of Simulation Experiments ....................... 55
  4.2.1 Frequency of Occurrence .......................... 55
  4.2.2 Query Types Studied .............................. 56
4.3 Experimental Results ................................. 56
4.4 Critical Analysis of The Results ...................... 59
5 CONCLUSIONS ............................................. 64
A APPENDIX A ............................................. 66
B APPENDIX B ............................................. 67
C APPENDIX C ............................................. 68
D APPENDIX D ............................................. 69
E APPENDIX E ............................................. 70
E.1 Details of Example 3.4 ................................. 70
BIBLIOGRAPHY ................................................ 77
F VITA AUCTORIS .......................................... 79
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 2.1</td>
<td>Join query graph $G$</td>
<td>9</td>
</tr>
<tr>
<td>Figure 2.2</td>
<td>Subquery graph $G_S$</td>
<td>10</td>
</tr>
<tr>
<td>Figure 3.1</td>
<td>Subquery graphs: S1, S2 and S3</td>
<td>42</td>
</tr>
<tr>
<td>Figure 3.2</td>
<td>Schedules for relations in $G$</td>
<td>43</td>
</tr>
<tr>
<td>Figure 3.3</td>
<td>Schedules for $R_0$ and $R_1$</td>
<td>45</td>
</tr>
<tr>
<td>Figure 3.4</td>
<td>Schedules for $R_2$, $R_3$ and $R_4$</td>
<td>46</td>
</tr>
<tr>
<td>Figure 4.1</td>
<td>Queries with selectivity $0.9 - 1.0$</td>
<td>60</td>
</tr>
<tr>
<td>Figure 4.2</td>
<td>Queries with selectivity $0.6 - 1.0$</td>
<td>61</td>
</tr>
<tr>
<td>Figure 4.3</td>
<td>Queries with different connectivity</td>
<td>62</td>
</tr>
<tr>
<td>Figure 4.4</td>
<td>Queries with different number of tuples in relations</td>
<td>63</td>
</tr>
<tr>
<td>Figure E.1</td>
<td>Subquery graphs: S1, S2 and S3</td>
<td>71</td>
</tr>
<tr>
<td>Figure E.2</td>
<td>Schedules for relations in $G$</td>
<td>72</td>
</tr>
<tr>
<td>Figure E.3</td>
<td>Schedules for $R_0$ and $R_1$</td>
<td>74</td>
</tr>
<tr>
<td>Figure E.4</td>
<td>Schedules for $R_2$, $R_3$ and $R_4$</td>
<td>76</td>
</tr>
</tbody>
</table>
List of Tables

Table 2.1  Domain values ............................................. 14
Table 2.2  Database statistics ....................................... 15
Table 3.1  Domain values ............................................. 32
Table 3.2  Database statistics ....................................... 32
Table 3.3  Domain values ............................................. 42
Table 3.4  The cost and benefit analysis for each cycle . . . 44
Table 4.1  Frequency of occurrence .............................. 55
Table 4.2  Query types ................................................ 56
Table 4.3  Experimental results .................................... 58
Table 4.4  Improvement of AHY and MJ algorithms .......... 59
Table E.1  Database statistics ....................................... 75
Chapter 1 INTRODUCTION

Distributed database system technology is one of the major recent developments in the database system area. Many "first generation" commercial distributed database products have been released. It has been claimed that within the next ten years, centralized database managers will be an "antique curiosity" and most organizations will move toward distributed database managers [Sto88].

Distributed query processing is one of the important topics in distributed database systems [OV91b]. A distributed query processor automatically translates any high-level query for a distributed database (which is seen as a single central database by the users) into an efficient low-level sequence of database operations. This sequence of operations is later executed on the local databases. In this thesis we study query optimization in distributed relational databases.

There are three phases to process a distributed query [YC84]:

1. **Local processing phase**
   This involves all local processing such as selections and projections.

2. **Reduction phase**
   A sequence of database operations is used to reduce the sizes of relations.
3. **Final processing phase**

All resulting relations are sent to the site where the final query processing is performed.

Significant research efforts have been focused on the problem of reducing the amount of data transmission required for phases 2 and 3. To solve the problem, most processing strategies produce schedules of database operations where semijoin operations precede the join operations in order to reduce the cost of data communication. This approach is called the semijoin strategy [AHY83, HY79, YC83, BC81, Seg86]. One of these strategies has been implemented in SDD-1 [BGW+81]. In some types of communication networks and database environments, using semijoin is not better than using join by itself, so some strategies perform the join operations directly, without semijoins. These are called join strategies [SAC+79, ESW78, Won83]. Some well-known systems implementing join strategies are distributed INGRES [ESW78] and System R* [LMH+85].

Most query processing strategies are static. The size of intermediate results is estimated before the query is executed, and it is assumed that the estimates remain unchanged throughout the query execution. Sometimes these strategies may be far from optimal because the size estimates may turn out to be inaccurate during the execution phase. However, little work has been done in the validation
of the optimization algorithms [Sel89].

1.1 The Thesis Statement and Topics to Be Investigated

We have investigated a new operation for distributed databases — the multi-attribute semijoin (MASJ) operation. The objective of our study is the following: investigate whether the multi-attribute semijoin (MASJ) operation may be applied to improve semijoin strategies in distributed query processing.

In order to establish that the multi-attribute semijoin operation is useful and interesting, we have attempted to study the following questions:

☑ When is it profitable to apply the multi-attribute semijoin operation?
☑ In what percentage of queries will this operation be profitable?
☑ How can this operation be incorporated into a query processing strategy?
☑ What improvements can be expected if the operation is combined with a standard query processing strategy based on the semijoin operation?

To answer these questions, we have studied properties of the MASJ operation and have incorporated this operation into the AHY Algorithm — a well known heuristic for distributed databases [AHY83]. We have carried out experiments where the sequence of database operations generated by our heuristic has been actually carried out and the actual communication cost using our heuristic has been compared to the actual cost using the AHY heuristic for query processing.
1.2 Organization of The Thesis

This thesis has five chapters. Chapter 2 introduces some important concepts and notations of query processing in distributed database system and contains a survey of previous work related to this thesis. Chapter 3 gives a detailed description of the MASJ operation, properties of this operation and presents, in detail, our algorithm, which we call the MJ Algorithm. Examples of this algorithm are also given to show how the MJ Algorithm works. Chapter 4 discusses the design of the test bed of queries, our evaluation method and our experimental results. Chapter 5 concludes the thesis.
Chapter 2  REVIEW OF LITERATURE

2.1 Distributed Database System

A Distributed Database Management System (DDMS) is used to store and manipulate information at widely dispersed locations [Ull88]. A DDMS has two components:

- A database management system to store, retrieve and manipulate data.
- A computer communication network to allow different sites of the system to communicate with one another.

Different architectures for database systems have been proposed (e.g. relational, hierarchical, network and object-oriented) [Dat95].

Communication networks may be classified using a number of criteria — network architecture (e.g. star, ring, hierarchical, mesh), mode of communication (broadcast, multicast and point-to-point), and geographic distribution (local area, wide area and metropolitan area) [Tan81].

Some advantages of distributed databases are as follows:
1. Each site in a DDMS is autonomous. This means all operations at a site are controlled by the site; local data is owned and managed locally, so that if a site is not operational, other sites remain functional.

2. The same data may be stored at more than one site so that, if a crash occurs at one of the sites, or a communication link fails and makes one or more sites inaccessible, the data may still be accessed from another site through another communication link. This possibility makes DDMSs more reliable.

3. It is much easier to increase the database size by adding another site.

4. Users can share data stored at different sites.

There are however, some inherent problems with DDMS [OV91b] as discussed below:

1. DDMSs are more complex than centralized databases, since we have to consider data communication, concurrence and synchronization of operations at different sites.

2. It requires additional hardware and also needs more complex software. In other words, hardware and software for DDMS are more expensive than that for centralized databases.

3. Because the data is transmitted over networks, there can be serious problems in maintaining adequate security over the communication network.
In a relational DDMS, it is possible that parts of the same relation are stored at different sites. In such cases, we say that the relation is fragmented [YC84]. Each part of a relation is called a fragment. A relation is horizontally fragmented if each fragment is a subset of all tuples in the relation. A relation is vertically fragmented if the attributes of a fragment is a subset of all attributes in the relation.

2.2 Distributed Query Processing and Optimization

Distributed query processing involves carrying out a sequence of database operations and data communications over the network. After the DDMS accepts a query, it has to generate several messages to process the query. In a simple query processing strategy, the following messages are needed:

1. A message is sent from the query site to each site where the data pertinent to the query is stored. This message requests information about the stored relations after all local processing has been carried out.

2. Each site receiving the above message sends a message back to give the requested information.

3. After formulating a strategy for processing the query efficiently, the query site sends messages to participating sites requesting a sequence of data communications and database operations. After the sequence is carried out, the resulting relations will be sent back from one or more of the participating sites to the query site.
To process a query, there are many possible processing strategies with widely varying communication/processing costs and delays. To minimize the cost or delay, it is crucial to find an efficient strategy for any given distributed query. *Query optimization* is the process of finding an efficient strategy.

Ideally, query optimization attempts to choose the best strategy from all possible query processing strategies. In practical terms, the optimal strategy may be too time consuming to evaluate [WC93]. Researchers therefore try to find heuristics to get a “near optimal” query processing strategy.

### 2.3 Cost Measures

When optimizing query processing, we have to determine what we wish to optimize — the total cost or the response time to process the query. The response time to process the query is the time from the initiation of the query to the time when the answer is produced. In a distributed database system, the total cost to process the query has two components — the local cost and the communication cost. The local cost includes the CPU and the I/O cost. The CPU cost is incurred when the CPU performs operations on data in main memory. The I/O cost is the time for disk input/output operations. The local cost can be minimized by reducing the number of I/O operations through fast access methods to the data and efficient use of main memory or disk. The communication cost is the cost of transmitting the data from one site to another on the communication network. In [YC84], the data
transmission cost between two sites is defined as a linear function: \( C_0 + C_1 \times X \), where \( C_0 \) is the start-up cost of initiating the transmission, \( C_1 \) is the cost coefficient associated with the communication of one unit of data and \( X \) is the amount of data (usually measured in number of units) to be communicated. For geographically dispersed computer networks, communication cost is normally the dominant cost.

### 2.4 Query Graph

A query \( Q \) can be expressed as a join query graph. A join query graph is denoted by the graph \( G = (V, E) \), where \( V \) is the set of nodes and \( E \) is the set of edges in the graph. The set of labels of all edges in \( E \) is denoted by \( L \). Node \( i \) in graph \( G \) corresponds to the site \( i \) of the network where the relation \( R_i \) is stored. The label of an edge is the name of the joining attribute of two relations so that an edge between nodes \( i \) and \( j \) with a label \( M \) denotes that there is an equijoin involving attribute \( M \) of \( R_i \) and \( R_j \). \( G_S = (V_S, E_S) \) is a subquery graph of \( G \) if \( V_S \subseteq V \), \( E_S \subseteq E \) and \( G_S \) is connected.

Figure 2.1 shows a join query graph \( G = (V, E) \), where the set of nodes

![Figure 2.1 Join query graph G](image)
REVIEW OF LITERATURE

$V = \{0, 1, 2, 3, 4\}$ and the set of labels $L = \{A, B, C, D\}$. Figure 2 shows the subquery graph $G_S = (V_S, E_S)$ of $G$. Here the set of nodes $V_S = \{1, 2, 3\}$ and the set of labels $L_S = \{A, B, C\}$.

2.5 Distributed Database Operations

We consider queries based on the SPJ model [Ull88] so that only select, project, and join operations are required. We view a relation $R$ as a two-dimensional table where each row of the table is a tuple in relation $R$ and each column is an attribute in $R$. Let $X$ is the set of all attributes in $R$, the operations on $R$ are defined as follows:

**Projection:**

The projection operation involves the relation $R$ and a set of attributes $T$, $T \subseteq X$. The operation consists of discarding all columns of $R$ that are not in $T$, and eliminating duplicate rows. The projection of relation $R$ on a set of attributes $T$ is denoted by $R(T)$.

**Selection:** The selection operation involves the relation $R$ and a condition involving one or more attributes of the relation and constant(s). The operation
produces the new relation with the same set of attributes and containing tuples that satisfy the specified condition.

A very common usage of this operation is to extract tuples with attribute \( A \) value equal to a specified constant in relation \( R \) (operators can also be "\( > \)", "\( < \)” and so on). In such cases, the operation is denoted by \( R. A = \text{the specified constant} \). Selection is obtained by choosing all tuples of \( R \) whose attribute \( A \) values are equal to the specified constant.

**Join:** The join operation involves two relations \( R_i \) and \( R_j \) and a set of attributes \( A \) such that \( A \subseteq (Y \cap Z) \) where \( Y \) and \( Z \) is the set of all attributes in \( R_i \) and \( R_j \). This operation is denoted by \( R_i.A \Rightarrow R_j.A \). The set of attributes \( A \) is called the joining attribute(s) of \( R_i \) and \( R_j \).

The join is obtained by concatenating each row of \( R_i \) with each row of \( R_j \) whenever the values of attributes in \( A \) in the two rows are equal and eliminating one set of values of attributes in \( A \). One commonly used join operation is the natural join, where \( A = (Y \cap Z) \). We will use the notation \( R_i \Rightarrow R_j \) to denote a natural join of relations \( R_i \) and \( R_j \). In this paper, if two relations in a query have the same attribute name, there is a join predicate between them.

**Semijoin:** The semijoin from relation \( R_j \) to relation \( R_i \) on attribute \( A \) is denoted by \( R_j - A \rightarrow R_i \), where \( R_j \) is the sending relation, \( R_i \) is the reduced relation, and \( A \) is the joining attribute. We use \( R_j \rightarrow R_i \) to represent
\( R_j - A \rightarrow R_i \) if there is no need to identify the attribute.

The semijoin can be obtained by joining \( R_i \) and \( R_j \) on attribute \( A \), then projecting the resulting relation on the all attributes of \( R_i \).

### 2.6 Estimating The Effect of Database Operations

One of the main factors affecting the performance of a query processing strategy is the sizes of the intermediate relations produced during the query execution phase. Estimating the communication cost to send a stored relation to another site is easy since the size of the relation is known. The situation becomes more complicated when we need to estimate the communication cost to send some intermediate relation to another site since it is necessary to estimate the sizes of these intermediate relations. This estimation is based on statistical information about the relations involved in the query and formulae used to predict the cardinalities of the relations obtained from a sequence of database operations. There is a direct trade-off between the precision of the statistical information and the cost of calculating such information [PSC84].

The estimation formulae used in this thesis (all notations can be found in Appendix A) are given below:

- **Formula to calculate the selectivity factor** \( p \) **of an attribute** \( A \) **in relation** \( R \):
Formulae to estimate the cardinality of an attribute after a semijoin:

Let relation $R_i$ and $R_j$ both have attribute $A$ and let $R_j$ have attribute $B$ which does not appear in $R_i$. After a semijoin operation $R_i \ - A \ ightarrow R_j$ and get $R'_j$, the number of distinct values of attributes $A$ and $B$ in $R'_j$ will, in general, change. Let $|R_j| = n$ and let $|R_j(B)| = m$. Let $|R'_j| = k$.

i) $|R'_j(A)|$ can be estimated as follows:

$$|R'_j(A)| = |R_j(A)| \times \frac{k}{n}$$

(2.2)

ii) $|R'_j(B)|$ can be estimated as follows [CY92]:

$$|R'_j(B)| = m \left[ 1 - \prod_{i=1}^{k} \left( \frac{n(m-1) - i + 1}{n - i + 1} \right) \right]$$

(2.3)

Equation (2.3) can be approximated as shown below [CY92]:

$$|R'_j(B)| = \begin{cases} m & m < \frac{k}{2} \\ k \frac{m+k}{3} & k < \frac{m}{2} \\ \text{otherwise} & \end{cases}$$

(2.4)

Formula to estimate the cardinality of $R_j \Rightarrow R_i$, $R_j \subseteq R_i(T)$, $T \subseteq X$, where $X$ is a set of attributes in $R_i$:

Let $R_i$ have attributes $A$, $B$, and $C$. $|R_i| = n$. $|R_i(A)| = m1$. $|R_i(B)| = m2$. $|R_i(C)| = m3$. $|R_i(T)| = m5$. $|R_j| = n$. $|R_j(A)| = m5$.
Let $R_j$ have attributes $A$ and $B$. $|R_j| = k$ and $R_j \subseteq R_i(A, B)$. Then

$$|R_j \Rightarrow R_i| = k + \frac{(n - k) \times k}{m_1 \times m_2}$$ \hspace{1cm} (2.5)

**Formula to estimate the cardinality of the result of a sequence of joins**

In [CY92], the following method is proposed to estimate the number of tuples in the resulting relation after a sequence of join operations:

Let $G = (V, E)$ be a join query graph. $G_B = (V_B, E_B)$ is a connected subgraph of $G$.

Let $R_1, R_2, \ldots, R_p$ be the relations corresponding to nodes in $V_B$ and let $A_1, A_2, \ldots, A_q$ be the distinct attributes associated with edges in $E_B$.

Let $m_i$ be the number of different nodes (relations) that the edges with attribute $A_i$ are incident to.

Let $R^*$ be the relation resulting from the join operations among relations in $G_B$, then

$$|R^*| = \frac{\prod_{i=1}^{p} |R_i|}{\prod_{i=1}^{q} |A_i|^{m_i-1}}$$ \hspace{1cm} (2.6)

**Example 2.1**

We consider the subquery graph $G_S = (V_S, E_S)$ (Figure 2.2) of query graph $G$ with database statistics as shown in Table 2.1 and 2.2. We now discuss how to use the formulae (2.1) — (2.6) given above.

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>F</td>
<td>G</td>
</tr>
<tr>
<td>1X1</td>
<td>580</td>
<td>559</td>
<td>575</td>
<td>551</td>
<td>550</td>
<td>550</td>
</tr>
</tbody>
</table>

Table 2.1 Domain values
1. **Formula (2.1):**

The selectivity factor $p$ of attribute $A$ in relation $R_1$ is calculated as follows:

$$\frac{|R_1(A)|}{|A|} = \frac{368}{580} = 0.63$$

2. **Formula (2.2):**

Assuming that the number of tuples in $R'_2$ became 3800 after the semijoin operation $R_1 \bowtie B \rightarrow R_2$. The cardinality of the join attribute $R'_2(B)$ can be estimated as follows:

$$|R_2(B)| \times \frac{|R'_2|}{|R_2|} = 505 \times \frac{3800}{5990} = 320$$

3. **Formula (2.4):**

Assuming that the number of tuples in $R'_2$ becomes 3800 after the semijoin
operation $R_1 - B \rightarrow R_2$, since $|R_2(C)| < |R'_2|/2$, the cardinality of the nonjoin attribute $|R'_2(C)|$ is estimated to be the same as $|R_2(C)|$ which is 465.

4. **Formula (2.6):**

When we join $R_1$ and $R_2$, the number of tuples in the resulting relation can be estimated as follows:

$$\frac{|R_1| \times |R_2| \times |R_3|}{|A| \times |B| \times |C|} = \frac{5680 \times 5990 \times 5140}{580 \times 559 \times 575} = 938$$

5. **Formula (2.5):**

The number of tuples in the relation $R' = R_1(A, B) \Rightarrow R_2 \Rightarrow R_3$ is 938 as discussed above. $R'$ should include attribute $A, B, C, G$ and $H$. If we project $R'$ on attribute $A$ and $B$ and join it with relation $R_1$, then the number of tuples in the new resulting relation can be estimated as follows:

$$|R'| \frac{(|R_1| - |R'|) \times |R'|}{|R_1(A)| \times |R_1(B)|} = \frac{(5680 - 938) \times 938}{368 \times 495} = 962$$

### 2.7 Query Processing Strategies

The semijoin operation has been established as a useful mechanism to reduce the amount of transmitted data when processing queries in distributed databases. Theoretical work on the semijoin operation can be found in [BC81]. Although the use of semijoins is a valuable tool, it is not always superior to the use of joins only. First, for some networks, the number of messages exchanged rather than the amount of data transferred may be the dominating factor. Second, most semijoin
strategies don’t consider local processing costs, but sometimes it can be very high using semijoin operations. Third, although semijoins can be executed in parallel, the minimization of response time using semijoins is complicated [YC84].

2.7.1 Semijoin Strategies

In this part, we illustrate two query optimization algorithms: SDD-1 and AHY. Both of them are devised for wide area point-to-point communication networks and employ semijoin operations. The cost of a semijoin $R_1 - A \rightarrow R_2$ is defined to be the cost of transferring $R_1(A)$ from the site containing relation $R_1$ to the site containing relation $R_2$. The benefit of the semijoin is the reduction in the size of $R_2$ as a result of the operation. A semijoin is profitable if its cost is less than its benefit.

The SDD-1 Strategy [BGW+81]

The SDD-1 is a distributed relational database system developed by the Computer Corporation of America. Users interact with SDD-1 by submitting queries coded in a high level procedural language called Datalanguage. The SDD-1 algorithm has three main steps [BGW+81]:

Step 1: Map a Datalanguage query $Q$ into relational calculus form (called an envelope) that specifies the superset of the database that is required to answer $Q$. 
Step 2: Construct a reducer P, which contains a sequence of relational operations. Select a site S such that the cost of computing P and moving the result to S is minimum over all reducers and sites.

Step 3: Execute Q at S using the data assembled by step 2.

The query optimization strategy used in step 2 is derived from an earlier method called the "hill-climbing" algorithm, which has the distinction of being the first distributed query processing strategy. The strategy assumes no data replication or fragmentation and minimizes both the total cost and the response time.

The SDD-I strategy accepts a query graph and the associated statistics for each relation as its input. It produces an execution strategy as its output. It needs four phases to process a distributed query:

1. **Initialization**: Carry out all local processing at each site, modify statistics to reflect the effect of local processing and generate a set of beneficial semijoins (BS).

2. **Selection of beneficial semijoins**: Select a beneficial semijoin from BS by iteratively choosing the most beneficial semijoin BBS and modifying the database statistics and BS. The interactive phase terminates when all semijoins in BS have been appended to the execution strategy (ES). The order in which the semijoins are appended to ES will be the order in which the semijoins
are executed.

3. **Assembly site selection:** For each candidate site, evaluate the cost of transferring all the required data to it. Select the site with the least cost as the assembly site.

4. **Postoptimization:** Generated BS is based on the assumption that relations can be transmitted to another site. After the assembly site is chosen, the data stored at the assembly site need not be communicated. Thus semijoins for the relation at assembly site is unnecessary. Remove unnecessary semijoins from ES.

**The AHY Strategy [AHY83]**

Algorithm GENERAL [AHY83] includes processing strategies for complex queries. It uses semijoin operations to reduce the data transmission cost. This algorithm generates a near optimal schedule of SPJ semijoin operations to process any SPJ query.

There are three versions of Algorithm GENERAL: one for minimizing response time and two for minimizing total time.

**Algorithm GENERAL:**

**Step 1:** Execute all initial local processing to reduce the amount of data to be transmitted.
Step 2: Generate candidate relation schedules.

A. To minimize response time, apply Algorithm PARALLEL to each simple query [HY79].

B. To minimize total time, apply Algorithm SERIAL to each simple query [HY79].

Step 3: Integrate the candidate schedules. After step 2, all candidate schedules are saved. In this step, those candidate schedules are integrated to form a processing schedule for each relation $R_i$. Procedure RESPONSE is for minimizing response time. Procedure TOTAL and procedure COLLECTIVE are for minimizing total time.

**Procedure RESPONSE:**

1. *Candidate schedule ordering.* Sort the candidate schedules on joining attribute $d_{ij}$ in ascending order of arrival time for each relation $R_i$.

2. *Schedule integration.* For each candidate schedule $l$ (the $l$th candidate schedule for $R_i$), create an integrated schedule for $R_i$ that consists of the parallel transmission of candidate schedule $l$ and all candidate schedules $k$ ($k<l$). Choose the integrated schedule which has the minimum response time.

**Procedure TOTAL:**

The version of total time is used in a multi-processing environment.
1. **Adding candidate schedules.** Check each candidate schedule \( l \) of \( R_i \), if this schedule contains a transmission of a joining attribute \( d_{ij} \) of \( R_i \), add another candidate schedule which is the same as candidate schedule \( l \) except that the transmission of \( d_{ij} \) is deleted.

2. **Select the best candidate schedule.** Choose the candidate schedule for each joining attribute \( d_{ij} \) of each \( R_i \) which minimizes the total time for transmitting \( R_i \) if only the joining attributes which can be joined with \( d_{ij} \) are considered.

3. **Candidate schedule ordering.** Sort \( \text{BEST}_{ij} \) (selected schedule for relation \( R_i \) considering joining attribute \( d_{ij} \)) on joining attribute \( d_{ij} \), \( j = 1, 2, \ldots, \sigma \). so that \( \text{ART}_{il} + C(s_i \times \text{SLT}_{il}) \leq \ldots \leq \text{ART}_{i\sigma} + C(s_i \times \text{SLT}_{i\sigma}) \). (\( \text{SLT}_{ij} \) is the accumulated attribute selectivity of the \( \text{BEST}_{ij} \) candidate schedule into \( R_i \), \( \text{ART}_{ij} \) is the arrival time of the \( \text{BEST}_{ij} \) schedule).

4. **Schedule integration.** Create an integrated schedule to \( R_i \) for each \( \text{BEST}_{ij} \) in ascending order of \( j \). The schedule consists of the parallel transmission of candidate schedule \( \text{BEST}_{ij} \) and all schedule \( \text{BEST}_{ik} \) \((k<j)\). Choose the integrated schedule which results in the minimum total time value

\[
TOTT_i = \sum_{k=1}^{j} \left[ \text{ART}_{ik} + C \left( s_i \prod_{k=1}^{j} \text{SLT}_{ik} \right) \right] \tag{2.7}
\]

**Procedure COLLECTIVE:**

Procedure TOTAL is not optimal, because it does not consider the existence of
redundant data transmissions in separate relation schedules. Procedure COLLECTIVE is used to handle this problem.

1. **Select candidate schedule.** Choose the minimum cost candidate schedule for each joining attribute $d_{ij}$ of each $R_i$, which contains the transmission of all components of attribute $j$ that selectivity is less than 1.

2. **Build processing strategy.** Use a removal heuristic to remove the most costly data transmission to derive a new strategy. Compare the total time cost of the new and old strategies, choose the strategy which costs less, then continue checking until no cost benefit can be obtained.

**Step 4.** Remove schedule redundancies. Some schedules for relations have been transmitted in the schedule of another relation. So the last step is to eliminate these relation schedules.

### 2.7.2 Join Strategies

**Distributed INGRES Strategy [ESW78]**

Distributed INGRES is a distributed relational database system and extended from centralized INGRES. In this distributed environment, the relations are allowed to be fragmented. The communication network can be either point-to-point or broadcast. In a point-to-point communication network, the cost is to send data from any site to any other site. In a broadcast communication network, the cost of sending data from one site to all sites is the same as that of sending the same
data from one site to any other single site. The query processing algorithm of Distributed INGRES is derived from the algorithm used in the centralized INGRES and is a dynamic strategy. It considers a combination of both the communication cost and the response time. Users interact with the database through the non-procedural query language QUEL.

The query processing algorithm accepts

1. A query expressed in tuple relational calculus.
2. The type of network.
3. The location and size of each fragment.

as input. Then it processes the query as follows:

1. Apply detachment operations [WY76] to the query. In this step, the query is decomposed into single variable subqueries (only one relation is involved in these subqueries) and irreducible subqueries (a query is irreducible only if none of its variable is a joining variable).
2. Execute all single-variable queries locally.
3. Apply the reduction algorithm [WY76] to the original query. In this step, according to the sizes of fragments, the irreducible subqueries are put into a certain sequential order, which can replace the original query.
4. Choose a irreducible subquery, which involves the smallest fragment in the list of irreducible subqueries. Determine all fragments which have to be
transferred and the processing site for the irreducible subquery.

5. Move the selected fragments to the processing site.

6. Execute the irreducible subquery at the processing site.

7. If the list of irreducible subqueries is not empty, go back to step 4. Otherwise, the algorithm terminates.

**R* Strategy [LMH+85]**

R* is a distributed relational database management system which evolved from System R, a centralized relational database management system. The SQL language is used to express a query in R*. The objectives of R* are:

- **Single-System Image:**
  
  Allow users to access the database without knowing the data is spread over different sites.

- **Site Autonomy:**
  
  a. To allow each site to operate on its local data, even if there are communication failures in other sites.

  b. To allow data sharing between two sites only if the DBAs in both sites agree to do so.

  c. To allow different versions of R* at different sites, so that any site can upgrade its R* version independently.
Minimal Impact on Performance:

To locate data close to the site(s) whose users access it the most, so as to minimize the cost of remote data access and transfer.

In R*, a query which is expressed by an SQL data manipulation statement, is processed in two steps:

1. Preparation phase:
   After a non-procedural SQL statement is submitted at a single site, compile it into access structures, which are a sequence of parameters to R* run-time routines, for all participating sites.

2. Execution phase:
   This phase is called by the user's program. After the run-time manager of the routine is called, it follows those parameters that identify the correct path to access the databases and move data between sites.

The R* query algorithm is used in the preparation phase to choose the access path. It considers both the local cost (CPU and I/O cost) as well as the communication cost, and is executed at the site where the query is initiated.

R* generates a processing strategy for a query as follows:

1. For each relation in the query, find all possible access paths to access the relation, and choose the one that has the minimum cost as the access path for this relation.
2. For each order of relations in the query, build a strategy to do the joins and calculate the cost of the strategy. Finally, choose the order with the minimum cost.

3. For each site involved in the query, choose a local processing strategy to carry out the local processing.

The purpose of query optimization in R* is to decide on five major aspects of execution [LMH+85]:

1. **Order of Join:** Consider all possible orders in which the relations are joined, adding one relation at a time. The impact of associativity is ignored.

2. **Access Method:** If indices are defined on any join or ordering columns, then compare the cost of doing an index scan with that of doing a sequential scan of the relation (in physical order), possibly sort it into an order (for a join or ordering clause).

3. **Method of Join:** Choose the merge join method or the nested loop join method.

4. **Join Site:** For the join of two given relations at distinct sites, choose the way of moving data from the following possibilities:

   a. Move the first relation to the site of the second relation to join them there.
   
   b. Move the second relation to the site of the first relation and join them there.

26
c. Move both relations to a third site.

5. **Intersite Data Transfer Strategy**: Choose one of two methods for intersite data transfers:

a. *Ship-whole*. Ship the entire relation to the join site and store in a temporary relation before executing the join. If the join method is the merge join, the relation doesn't need to be stored, and the join site can process incoming tuples in a pipeline mode, as they arrive.

b. *Fetch-as-needed*. Scan the external relation sequentially. For each tuple, send the join value to the site of the internal relation, which selects the internal tuples matching the value and sends the selected tuples to the site of the external relation. This method is equivalent to the semijoin of the internal relation with each external tuple.
In this chapter, we introduce and discuss a new type of semijoin operation, which is called the Multi-attribute Semijoin (MASJ) operation. After discussing the properties of this operation, we present a query optimization procedure in which this operation is an additional step in the AHY Algorithm (total time version). We call it the MJ Algorithm. We assume that there is no data replication or fragmentation so that each site stores only one relation and each relation is stored only at one site. The network, which is employed to link all sites, is a point-to-point wide area network. Since communication cost dominates the local processing cost in geographically dispersed computer networks, we are only concerned with the communication cost as measured by the amount of data communicated on the network.

3.1 Notion of a Multi-attribute Semijoin (MASJ) Operation

The MASJ operation is a composite operation which reduces the number of tuples in a relation by a set of database operations. The multi-attribute semijoin operation is somewhat different from the well known semijoin operation, which we introduced in Chapter 2. We call this operation a semijoin operation because
each relation sends, to the operation site, only those attributes that will participate in the MASJ operation. In all cases, more than one attribute will be sent from each relation participating in the operation to the site where the operation is being carried out. When a MASJ operation is used to reduce a relation, the size of the relation is reduced not by eliminating values of attributes individually, but by eliminating the combinations of values of more than one attribute.

Let $S = (V_S, E_S)$ be a subquery graph of $G$, let node $j, j \in V_S$, be chosen to be the site of the operation. We will use $R_j$ to refer to the relation stored at site $j$. This is possible since only one relation is stored at one site. The size and the attributes of the relation stored at a site will change as semijoin and/or MASJ operations are carried out on the current relation stored at this site. The term $R_j$ always refers to the relation currently stored at site $j$.

We denote the MASJ operation for the subquery graph $S$ by $\otimes (j, S)$, where $j$ is the site of the operation. $\text{ATTR}(S)$ denotes the set of all edge labels in $S$, $\text{SCHEMA}(R_i)$ denotes the set of all attributes in $R_i$.

- **Definition of the MASJ operation**

The operation $\otimes (j, S)$ is expressed as follows:

\[
\text{For all } i, i \in V_S - \{j\} \text{ do} \\
R_i(\text{SCHEMA}(R_i) \cap \text{ATTR}(S)) \Rightarrow R_j
\]

---

1 Later on, in this section, we will discuss how we choose a promising subquery graph and a site of operation
MJ ALGORITHM USING MULTI-ATTRIBUTE SEMIJOIN (MASJ) OPERATION

- **Attributes in the result of a MASJ operation**

  We note that $R_i(SCHHEMA(R_i) \cap ATTR(S))$ may contain attributes which are not in $R_j$. After the operation $\otimes (j. S)$, the attributes stored in site $j$ are as follows:

  $$\left[ \bigcup_{i \in V_S - \{j\}} SCHHEMA(R_i) \cap ATTR(S) \right] \cup SCHHEMA(R_j) \quad (3.2)$$

- **The cost of the MASJ operation $\otimes (j. S)$**

  The cost of $\otimes (j. S)$ is the total amount of data which has to be sent from each site $i$, $i \in V_S - \{j\}$, to site $j$. We may estimate this cost using the following formula:

  $$\sum_{i \in V_S - \{j\}} \omega_{(SCHHEMA(R_i) \cap ATTR(S))} \times |R_i| \quad (3.3)$$

- **The benefit of the MASJ operation $\otimes (j. S)$**

  The operation $\otimes (j. S)$ may be viewed as a number of projections followed by a number of joins. The formula (2.6) may be used to calculate the number of tuples from this sequence of join operations. Details of this calculation are given in Appendix B. We use $M_{tuple}$ to stand for the number of tuples resulting from a MASJ operation. The benefit of $\otimes (j. S)$ in $j$ can be calculated as follows:

  $$\left( |R_j| - M_{tuple} \right) \times \omega_{SCHHEMA(R_j)} \quad (3.4)$$
MJ ALGORITHM USING MULTI-ATTRIBUTE SEMIJOIN (MASJ) OPERATION

- **The net benefit of the MASJ operation** \( \otimes (j, S) \)

  The net benefit of the MASJ operation is the difference between the benefit and the cost of the operation as defined above. If the net benefit is positive, we say that the \( \otimes (j, S) \) is beneficial.

**Example 3.1**

We consider the subquery graph \( S = (V_S, E_S) \) (Figure 2.2) of query graph \( G \) (Figure 2.1) with database statistics as given in Tables 3.1 and 3.2 where \( i \in V \) and \( X \in L \). We will now discuss how to carry out the MASJ operation on \( S \) at site 2. In other words, we will carry out the operation \( \otimes (2, S) \).

As noted in section 2.4, \( V_S = \{1, 2, 3\} \) and \( L_S = Attr(S) = \{A, B, C\} \). In site 2, the MASJ operation will be carried out using (3.1):

\[
i = 1:
R_1(A, B) \Rightarrow R_2
\]

Since \( SCHEMA(R_1) = \{A, B, D, F\} \). Thus \( SCHEMA(R_1) \cap Attr(S) = \{A, B, D, F\} \cap \{A, B, C\} = \{A, B\} \).

\[
i = 3:
R_3(A, C) \Rightarrow R_2
\]

Since \( SCHEMA(R_3) = \{A, C, H\} \). Thus \( SCHEMA(R_3) \cap Attr(S) = \{A, C, H\} \cap \{A, B, C\} = \{A, C\} \).

Here the order of the two database operations is immaterial.
The cost of $\otimes (2. S)$:

The cost of the operation is the sum of the cost to send $R_{1}(A, B)$ and the cost to send $R_{3}(A, C)$ to site 2. It is computed using formula (3.3)²

$$|R_{1}| \times \omega_{\{A,B\}} + |R_{3}| \times \omega_{\{A,C\}} = 17040 + 15420 = 32460$$

---

² Because it is difficult to estimate the size of a relation after projecting on more than one attribute, we choose the worst case, which the size of the relation wasn’t reduced.
The benefit of $\otimes(2, S)$:

The benefit of the operation is the reduction in the size of the relation stored at site 2. After the MASJ operation, the size of $R_2$ including the original attributes at site 2 may be computed using formula (3.4)

\[
\left( |R_2| - \frac{|R_1| \times |R_2| \times |R_3|}{|A| \times |B| \times |C|} \right) \times \omega_{\{B,C,G\}} = (5990 - 938) \times 7 = 5052 \times 7 = 35364
\]

The net benefit of the MASJ operations $35364 - 32460 = 2904$. Obviously, this operation is beneficial.

- **Procedure to determine the site of a MASJ operation**

  Now we discuss how we determine the site where a MASJ operation on a given subquery graph may be carried out most economically. The procedure is as follows:

  - Estimate the communication cost of sending out data from each site.
  - Choose the site which involves the maximum communication cost as the operation site.

  In other words, site $j$ is chosen to be the site of the operation $\otimes(j, S)$, if for all $i$, $i \in V_S - \{j\}$,

  \[
  \omega_{\text{SCHEMA}(R_j) \cap \text{ATTR}(S))} \times |R_j| \geq \omega_{\text{SCHEMA}(R_i) \cap \text{ATTR}(S))} \times |R_i|
  \]

  \[ (3.5) \]
Example 3.2

We will now show how the above procedure may be used to determine the site where the MASJ operation on subquery \( S \) as defined in example 3.1 may be carried out.

The cost to send \( R_1: |R_1| \times \omega_{\{A,B\}} = 5680 \times 3 = 17040 \)

The cost to send \( R_2: |R_2| \times \omega_{\{B,C\}} = 5990 \times 4 = 23960 \)

The cost to send \( R_3: |R_3| \times \omega_{\{A,C\}} = 5140 \times 3 = 15420 \)

This calculation shows that the communication cost to send the relevant attributes of \( R_2 \) to any other site is higher than that from any other site in \( S \). This is the reason why, in example 3.1, we chose site 2 to be the site of the MASJ operation for subquery graph \( S \).

3.2 Promising Subquery Graph and Schedule With MASJ Operation (MSCH)

Given a query graph, the number of possible subquery graphs can be enormous. It is therefore necessary to have a procedure to identify those subgraphs where a MASJ operation is likely to be useful.

Definition:

\( S = (V_S, E_S) \) is a **promising subquery graph** of query \( G \) if there is a MASJ operation \( \otimes (j, S) \), \( j \in V_S \), such that the net benefit of applying \( \otimes (j, S) \) to
reduce \( R_j \) is positive and is greater than the net benefit of reducing \( R_j \) using any sequence of semijoin operations.

**Definition:**

A subquery graph \( S = (V_S, E_S) \) is **more promising** than a subquery graph \( S_1 = (V_{S1}, E_{S1}) \) if there exists a schedule of database operations including the operation \( \otimes (j, S) \), for some \( j \in V_S \), which is expected to require less data communication than any other schedule of database operations which includes the operation \( \otimes (j, S1) \) and reduces \( R_j \) to the same extent.

**Theorem 1:**

If \( S = (V_S, E_S) \) is a promising subquery graph and there is a node \( n, n \in V_S \), such that all edges \( e \in E_S \) incident to \( n \) have the same label \( A \) so that they represent the same attribute, then \( S \) cannot be more promising than \( S_1 = (V_{S1}, E_{S1}) \), where \( V_{S1} = V_S - \{n\} \) and \( E_{S1} \) contains all the edges of \( E_S \) involving nodes in \( V_{S1} \).

**Proof:**

Let \( i \) be the site of \( \otimes (i, S) \). If \( \frac{|R_n(A)|}{|A|} \geq 1 \), then clearly \( S_1 \) is more promising.

If \( \frac{|R_n(A)|}{|A|} < 1 \), we have two cases to consider: \( n = i \) or \( n \neq i \). If \( n \neq i \), we choose any site \( k \) such that \( R_k \) is a relation with \( A \in SCHEMA(R_k) \).

The cost of \( R_n - A \rightarrow R_k \) is the same as the cost of sending \( R_n(A) \) to \( i \).

However, the operation \( R_n - A \rightarrow R_k \) reduces the number of tuples of \( R_k \) (since \( \frac{|R_n(A)|}{|A|} < 1 \)) and therefore reduces the cost of \( \otimes (i, S1) \). If \( n = i \), let
MJ ALGORITHM USING MULTI-ATTRIBUTE SEMIJOIN (MASJ) OPERATION

$j$ be the best site for MASJ operation on $SI$. The cost of performing $\otimes (j, S1)$ is less than that of $\otimes (i, S)$ since this does not involve any data communication from $R_j$. Since the only common attribute between $R_j'$ and $R_n$ is $A$, where $R_j' = \otimes (j, S1)$ we may now carry out the operation $R_j' - A \rightarrow R_n$ giving the same number of tuples at $i$. Since $j$ is the best site for the operation on $SI$, there is no site $k$, with a relation $R_k$ containing $A$ such that $|R_k(A)| > |R_j(SCHEMA(R_j) \cap ATTR(S1))|$. Thus the cost of the operation $R_j' - A \rightarrow R_n$ can not exceed the cost of $|R_j(SCHEMA(R_j) \cap ATTR(S))| \Rightarrow R_n$.

\textbf{Theorem 2:}

Each node in a promising subquery graph lies on one or more cycles.

\textbf{Proof:}

Each node in a promising subquery graph has at least two or more edges with different labels. If a node does not appear on a cycle, at least one node has only one edge. Since each node has two or more edges, each node must be on one or more cycles.

Since, in general, there are exponential number of cycles in a graph, determining all cycles is a problem of exponential complexity [Har69], and finding overlapping cycles is intractable. In the MJ Algorithm, heuristic $H$ identifies only
MJ ALGORITHM USING MULTI-ATTRIBUTE SEMIJ OIN (MASJ) OPERATION

a limited number of cycles. The heuristic ensures that each cycle has edges with
different labels in the query graph. The details of heuristic H are in Appendix C.

It is important to note that once a relation has been reduced substantially by a
MASJ operation, this relation may be used as an effective reducer to reduce other
relations. For instance, in example 3.1, we carried out the operation $\otimes (2. S)$
to reduce the relation $R_2$. After this operation, the relation at site 2 includes all
the join attributes appearing in the subquery graph $S$ (Figure 2). A secondary
advantage of a MASJ operation $\otimes (j. S)$ is that we can often use the result of
the operation at site $j$ to reduce one or more of the relations at other nodes of
the subquery graph $S$.

Example 3.3

In example 3.1, after $\otimes (2. S)$, the number of tuples at site 2 can be calculated
by using formula (2.6) as follows:

$$\frac{|R_1| \times |R_2| \times |R_3|}{|A| \times |B| \times |C|} = 938$$

The attributes in the relation at site 2 after this operation are $A$, $B$, $C$ and $D$
(obtained by applying formula (3.2)). Now we use $R_2$ to reduce $R_1$ and $R_3$.

The cost to send $R_2(A, B)$ to site 1 is calculated as $|R_2| \times \omega_{\{A, B\}} = 938 \times 3 = 2814$

The number of tuples in the reduced relation $R_3$ (formula (2.5)) is calculated as

$$|R_2| + \frac{|R_1| - |R_2| \times |R_2|}{|R_1(A)| \times |R_1(B)|} = 938 + \frac{(5680 - 938) \times 938}{368 \times 495} = 962$$
MJ ALGORITHM USING MULTI-ATTRIBUTE SEMIJOIN (MASJ) OPERATION

We note that the size of $R_i$ is reduced dramatically from 5680 to 962 as a result of this operation.

The cost to send $R_2(A, C)$ to site 3 is calculated as $|R_2| \times \omega_{\{A,C\}} = 938 \times 3 = 2814$

The number of tuples in the reduced relation $R_i$ (formula (2.5)) is given by

$$|R_2| + \frac{(|R_3| - |R_2|) \times |R_2|}{|R_3(A)| \times |R_3(C)|} = 938 + \frac{(5140 - 938) \times 938}{428 \times 425} = 960$$

Again we note that the size of $R_3$ is reduced dramatically.

After we find cycles in $G$ using heuristic H, we have to determine which cycle gives us the maximum benefit when we carry out a MASJ operation. We should consider not only the immediate benefit at the site of the operation (benefit of MASJ operation) but also the benefit from reducing other relations. We introduce the term MSCH (schedule with MASJ operation) to refer to the schedule of database operations which includes a MASJ operation and is followed by a number of projection and join operations to reduce some other relations in $S$. A MSCH including the MASJ operation $\otimes (j, S)$ will include the operation

$$R_j (SCHEMA(R_i) \cap ATTR(S)) \Rightarrow R_i, \ i \in V_S - \{j\},$$

if the cost of the operation is less than the benefit.
For instance, in example 3.3, the cost to send $R_2(A, B)$ to site 1 is 2814 and the benefit of reducing $R_I$ is $(5680 - 938) \times 3 = 14226$, so this operation to reduce $R_I$ is beneficial. The benefit of a MSCH is the sum of the benefits from reducing each relation in a MSCH. Similarly, the cost of a MSCH is the sum of the costs from reducing each relation in a MSCH. The net benefit of a MSCH is the difference between the benefit and the cost. For instance, the net benefit of the MSCH of $S$ in examples 3.1 and 3.3 is the sum of benefits minus the sum of the costs from all the database operations in MSCH on sites 1, 2 and 3.

The total sum of all benefits for the above MSCH is as follows:

$$(5680 - 962) \times 5 + (5990 - 938) \times 7 + (5140 - 960) \times 4 = 75674$$

The sum of all costs associated with the above MSCH is as follows:

$$2814 + 32460 + 2814 = 38088$$

In this case, the net benefit is positive.

Since, in general, for a given query graph, we have to look at a number of cycles, identified by heuristic $H$, we need a way to measure how promising a given cycle is. We define a factor $F$ for a cycle as follows:

$$F = \frac{\text{net benefit of MSCH}}{\text{number of nodes in subgraph}} \quad (3.6)$$

A cycle $S1$ is more beneficial than a cycle $S2$ if $F1$ of $S1$ is greater than $F2$ of $S2$. After identifying promising cycles using heuristic $H$, we determine which cycle
MJ ALGORITHM USING MULTI-ATTRIBUTE SEMIJOIN (MASJ) OPERATION

has the highest value for the factor F. This cycle is the best cycle for carrying out the MASJ operation.

3.3 The MJ Algorithm — Our Heuristic for Query Processing

The purpose of this study is not to propose a definitive strategy for query processing but to establish the viability of the MASJ operation. In order to investigate whether the MASJ operation is useful, we need to develop a strategy to process queries using this operation in addition to other known operations. We have chosen to augment the AHY Algorithm [AHY83] with the MASJ operation to study how much improvement we get.

3.3.1 The Steps of The MJ Algorithm

For each query, the MJ Algorithm generates a schedule of database operations \(QSC\). Initially \(QSC\) is empty. The steps to generate \(QSC\) are as follows:

Step 1. Find all candidate cycles

Identify each cycle in \(G\) such that successive edges in each cycle have different labels using heuristic \(H\).

Step 2. Determine the best cycle (\(BC\)), the site of the MASJ operation in \(BC\) and generate the MSCH schedule including the MASJ operation \(\otimes (j, BC)\).

A. Generate reducers, using the AHY algorithm, for relations in \(G\).
MJ ALGORITHM USING MULTI-ATTRIBUTE SEMIJOIN (MASJ) OPERATION

B. From all candidate cycles identified in step 1, determine the best cycle 
\((BC = (V_{BC}, E_{BC}))\) having the maximum value of factor \(F\).

C. Find the site \(j\) of the MASJ operation for \(BC\).

D. Generate MSCH, a schedule of database operations. MSCH always 
includes the multi-attribute semijoin operation \(\otimes (j, BC)\). In addi-
tion, it will include semijoin operations using the reducers obtained 
in step A, if such semijoin operations are beneficial.

E. Append to MSCH database operations for reducing the sizes of some 
relations in BC using the result of the MASJ operation.

F. Append MSCH to \(Q SCH\).

Step 3. **Generate new reducers to reduce the sizes of relations not appearing in BC**

Based on the new database information, generate new reducers using 
AHY for the remaining relations.

Step 4. **Check the remaining cycles in \(G\)**

Delete all cycles containing one or more nodes which appear in BC.

Go back to step 2 if there is at least one cycle. Otherwise, stop after 
appending all the reducers obtained in step 3 to \(Q SCH\).

3.3.2 Example 3.4

We give an example to illustrate the algorithm. To simplify discussions, we
MJ ALGORITHM USING MULTI-ATTRIBUTE SEMIJ OIN (MASJ) OPERATION

have not included details. Later on, we have described the algorithm in more detail and we have included the same example showing how each step is actually carried out in Appendix E.

We will use the query $G = (V, E)$ of example 3.1. To simplify our explanations we assume that all attributes are one byte long as shown in Table 3.3.

<table>
<thead>
<tr>
<th>X</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>X</td>
<td></td>
<td>580</td>
<td>559</td>
<td>575</td>
<td>551</td>
<td>550</td>
<td>550</td>
<td>550</td>
</tr>
<tr>
<td>$\omega(X)$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 3.3 Domain values

Step 1: Find all candidate cycles

If we apply heuristic H, we get the cycles shown in Figure 3.3.

Figure 3.1 Subquery graphs: $S1$, $S2$ and $S3$
Step 2: Determine the best cycle, the site of the MASJ operation in BC and generate the MASH schedule including a MASJ operation.

The AHY Algorithm generates a schedule for each relation in $G$ as shown in Figure 3.4. If we use these reducers to reduce all relations in $G$, the sizes of all

\[ \begin{align*}
    &d_1^A \rightarrow d_3^A \rightarrow R_0 \\
    &d_1^D \rightarrow 380 \\
    &d_0^A \rightarrow d_3^A \rightarrow R_1 \\
    &d_0^D \rightarrow 470 \rightarrow 8643 \\
    &d_2^B \rightarrow 505 \\
    &d_3^C \rightarrow 425 \rightarrow 11762 \\
    &d_1^B \rightarrow 495 \\
    &d_1^A \rightarrow d_0^A \rightarrow R_3 \\
    &d_2^C \rightarrow 465 \\
    &d_1^B \rightarrow d_2^B \rightarrow R_4 \\
    &d_2^B \rightarrow 448 \rightarrow 9248
\end{align*} \]

Figure 3.2 Schedules for relations in $G$
relations after reduction are as follows:

\[ |R_0| = 1918. \quad |R_1| = 2160. \quad |R_2| = 3921. \quad |R_3| = 1764. \quad |R_4| = 4624 \]

The cost and benefit analysis for each cycle is given in Table 3.4.

<table>
<thead>
<tr>
<th>Cycle</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
</tr>
</thead>
<tbody>
<tr>
<td>The benefit of MSCH</td>
<td>-17657</td>
<td>21858</td>
<td>16366</td>
</tr>
<tr>
<td>The cost of MSCH</td>
<td>11685</td>
<td>7851</td>
<td>3836</td>
</tr>
<tr>
<td>The net benefit of MSCH</td>
<td>-29342</td>
<td>14007</td>
<td>12530</td>
</tr>
<tr>
<td>$F$</td>
<td>-734</td>
<td>4669</td>
<td>6265</td>
</tr>
<tr>
<td>Comments</td>
<td>delete $S1$ since $F &lt; 0$</td>
<td>keep $S2$</td>
<td>keep $S3$</td>
</tr>
</tbody>
</table>

Table 3.4 The cost and benefit analysis for each cycle

Cycle $S3$ is the best cycle to carry out the first MASJ operation since $F$ of $S3$ is the greatest. The cost to communicate $R_0(A, D)$ is 3836 and the cost to communicate $R_I(A, D)$ is 4320. The best site to carry out this operation is site 1 since the cost is higher.

The MSCH of database operations is shown in Figure 3.3. It consists of the following sequence of operations:

1) A reducer generated by the AHY algorithm to reduce $R_0$
2) A MASJ operation at site 1 using the reduced relation $R_0$
3) A join of reduced relation $R_I(A, D)$ and $R_0$
This MSCH is appended to \textit{QSCH}, the schedule to process the query.

**Step 3:** Generate new reducers to reduce the size of relations not appearing in \textit{BC}.

The database operations carried out in step 2 have reduced the sizes of relations \( R_0 \) and \( R_1 \). We can now use these results to reduce the remaining relations. If we use the AHY algorithm with the current contents of relations \( R_0 \) and \( R_1 \), we get the new schedules for \( R_2, R_3, \) and \( R_4 \) as shown in Figure 3.4.
We note that these reducers are better than those reducers shown in Figure 3.4 for $R_2$, $R_3$, and $R_4$.

**Step 4: Check the remaining cycles in $G$**

In this example, $S_2$ is the only remaining cycle with a positive value for $F$, but $S_2$ includes some nodes which are also included in $S_3$, so no additional MASJ operation is feasible. The process stops after appending the three schedules to $QSCH$ as shown in Fig 3.4.

### 3.4 Details of The MJ Algorithm

In this section we will discuss, in detail, the steps of the MJ Algorithm. We will show how the steps given in example 3.4 have been actually carried out in
3.4.1 Details of Step 1

To identify all candidate cycles in $G$, we use heuristic $H$ to construct trees, which are similar to the algorithm to generate spanning trees [Har69], with the condition that, for any interior node $N \in V$ in the tree, the label of the edge from the parent($N$) to $N$ is different from the label of edge from $N$ to any of its children. We carry out a depth first search and check for cycles that pass through the root node. Details of heuristic $H$ are given in Appendix C. Heuristic $H$ guarantees that no duplicate cycle is generated. The complexity [Har69] of heuristic $H$ is $O(|V|^2 \cdot |E|)$.

3.4.2 Details of Step 2

Here we

- determine the best cycle (BC)
- find the site to carry out a MASJ operation in BC most economically
- generate a schedule MSCH of database operations including a MASJ operation

We assume that $n$ cycles are found in step 1, and that there are $m$ relations in cycle $S_i$. 

47
MJ ALGORITHM USING MULTI-ATTRIBUTE SEMIJOIN (MASJ) OPERATION

A. Apply the AHY Algorithm to the query $G$, and generate one schedule for each relation. (The term $SCH_j$ in the description below refers to the schedule of the relation $R_j$.)

B. For each candidate cycle $S_i$ ($i = 1, 2, ..., n$):

a. For all relations $R_j$ appearing in cycle $S_i$, calculate the lowest cost to communicate the attributes

\[
\text{SCHEMA}(R_j) \cap \text{ATTR}(S_i)
\]

in $R_j$ to any other site. Use the AHY reducer $SCH_j$ for $R_j$, found in step 1, to reduce, if beneficial, the cost of communication. Details are given in Appendix D.

b. Decide on the site of the MASJ operation using formula (3.5).

c. Calculate the factor $F$ using formula (3.6).

C. Choose as the best cycle (BC) the cycle that has the greatest value for factor $F$. If $F > 0$, the site with the greatest communication cost is the site of the MASJ operation. Let $bs$ be the site of the operation. If $F < 0$, then no cycle can be found.

D. If a cycle was found in step 3, generate a schedule $\text{MSCH}$ of database operations. The schedule contains the following:

- Schedule $SCH_j$, for each relation $R_j$ in BC, $j \neq bs$, where the application of $SCH_j$ was profitable in step Ba.
**MJ ALGORITHM USING MULTI-ATTRIBUTE SEMIJOIN (MASJ) OPERATION**

- MASJ operation $\otimes_{bs.} (BC)$.

- A join $R_{bs}(\text{SCHEMA}(R_j) \cap \text{ATTR}(Si)) \Rightarrow R_j$, for each relation $R_j$ in BC, whenever the cost of this operation is less than the benefit.

E. Append MSCH to $QSCH$.

### 3.4.3 Details of Step 3

Here we generate new reducers to reduce the size of relations not appearing in BC. We often see a dramatic reduction in the size of $R_j$. This means that it might be possible to use relations currently at nodes in BC to reduce the sizes of relations which are not in BC but which are adjacent nodes in BC. We use the new database information to reduce node $n$, $n \in V - V_S$.

In this step, based on new database statistics,

A. Update candidate schedules for each $R_n$, which are generated by step 2 of AHY Algorithm.

B. Go to step 3 of AHY Algorithm, which integrates the candidate schedules, to generate a new schedule for each $R_n$.

### 3.4.4 Details of Step 4

In this step we check the remaining cycles in $G$ and reject cycles that contain relations in BC.
MJ ALGORITHM USING MULTI-ATTRIBUTE SEMIJOIN (MASJ) OPERATION

A. Check each of the remaining cycles $S_r = (V_r, E_r)$ by the following procedure:

$$\text{If } (V_r \cap V_S = \emptyset)$$

keep the cycle

else

delete the cycle

B. Go back to step 2 if there is at least one remaining cycle. Otherwise, stop after appending all the schedules obtained in step 3 to $QSCH$. 
Chapter 4 EXPERIMENTS AND EVALUATION

To study whether the use of the MJ Algorithm leads to better performance, we carried out various experiments based on a large number of queries. In this chapter, we will discuss our objectives, present experimental results, and finally discuss the significance of these results.

4.1 Objectives of Our Test

To evaluate the MJ Algorithm, we designed different experiments with the following objectives:

1. Test the MJ Algorithm and the AHY Algorithm with a query set consisting of a wide variety of SPJ type queries.

2. For each query in the query set, estimate the cost for processing the query using each of the two algorithms. This estimate is based on statistical information about the database.

3. Determine the accuracy of the statistical estimation procedure, using actual database operations on synthetic databases.

4. Compare the performance of the MJ Algorithm with the AHY Algorithm.
4.1.1 Generating The Set of Queries

In a distributed database system, a SPJ query \( R_1 \Rightarrow R_2 \Rightarrow ... \Rightarrow R_n \) has to be processed taking into account the properties of the relations \( R_1, R_2, ..., R_n \) and their attributes. We need to decide which parameters are most important in distributed query processing since it is unrealistic to construct all types of queries by systematically modifying all parameters and test all of them.

We fixed the values of some of the parameters which, in our opinion, had minor influence on the performance of the MJ Algorithm and varied the remaining parameters which, we felt, had substantial influence on the query processing strategies.

We felt that the extent to which relations are connected to each other is likely to be significant. We use the term connectivity [Bea95] as defined below to quantify this. Let a query consist of \( n \) relations, each relation consists of 1 to \( m \) joining attributes. The connectivity is defined as follows:

\[
\text{connectivity} = \frac{\sum_{i=1}^{n} \text{(number of join attributes in } R_i)}{n \times m} \times 100\%
\]  

The parameters which we expect to have a significant effect on performance are as follows:

- The number of relations involved in the query
- The number of join attributes in a relation
EXPERIMENTS AND EVALUATION

- The maximum ratio of the number of tuples in a relation to the domain size of the join attribute that appears in the relation.

- The selectivity of each attribute in a relation (ratio of the number of distinct values of the attribute to the domain size of the attribute in the relation).

- The connectivity of the relations.

The parameters which are not expected to have a major effect on performance are given below:

- Width of an attribute.

- Number of nonjoin attributes.

In order to join all relations of a query, we ensured that these relations are connected. The test parameters are given below:

1. Each query consisted of 3 or 6 relations, and each relation consisted of 1 to 4 joining attributes.
2. The domain size of each joining attribute varied between 500 and 600.
3. The width of each attribute was 1 (i.e. $\omega(A_j) = 1$)
4. The number of tuples in each relation varied between 1500 and 2000, or between 5000 and 6000.
5. The selectivity varied between 0.6 and 1, or between 0.9 and 1.
6. We chose connectivity: 40%-50% or 60%-70%.
7. Each relation had one attribute which was required at the query site and didn’t appear in any other relation.

We used the query generator written by Mr. T. Bealor [Bea95] to generate all queries.

4.1.2 The Test Database [Bea95]

As mentioned earlier, we have used two methods to compare the MJ and the AHY strategies. One way is to take the schedule for the query generated by a strategy and determine its cost using statistical information. The second and more accurate way to obtain the actual cost of executing schedules is to simulate the execution of a schedule by carrying out the database operations in the schedule on a synthetic database. To construct the relations participating in a query we generated the tuples in each relation using following the rules:

- Only positive consecutive integer values are allowed for the values of each attribute. For example if $|A|$ is 1000, then the possible values of attribute $A$ are 0 — 999.
- To obtain values for $R(A)$, we randomly selected values from the domain of attribute $A$. If $R(A)$ has selectivity 0.5 and $|A| = 1000$, we randomly selected 500 distinct values from the domain of $A$ to constitute the actual values for $R(A)$. 
EXPERIMENTS AND EVALUATION

For each query, we generated relations using these rules to test the algorithms. The statistical information from the query generator was used by the relation constructor and was coded in C\(^3\) [Bea95].

4.2 Results of Simulation Experiments

4.2.1 Frequency of Occurrence

First, we needed to determine what percentage of queries have cycles on which MASJ operations may be carried out. To answer this question, we tested seven types of queries with connectivity ranging from 50\% to 100\%. For each type of query, we randomly generated 100 queries. The results are shown in Table 4.1. This result establishes that it is indeed useful to study this operation since the scenario where this type of operation is useful occurs very frequently.

<table>
<thead>
<tr>
<th>Number of relations</th>
<th>Number of attributes</th>
<th>frequency (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1 - 3</td>
<td>100</td>
</tr>
<tr>
<td>4</td>
<td>1 - 3</td>
<td>84</td>
</tr>
<tr>
<td>4</td>
<td>1 - 4</td>
<td>100</td>
</tr>
<tr>
<td>5</td>
<td>1 - 3</td>
<td>86</td>
</tr>
<tr>
<td>5</td>
<td>1 - 4</td>
<td>96</td>
</tr>
<tr>
<td>6</td>
<td>1 - 3</td>
<td>93</td>
</tr>
<tr>
<td>6</td>
<td>1 - 4</td>
<td>100</td>
</tr>
</tbody>
</table>

Table 4.1 Frequency of occurrence

\(^3\) The query generator and the relation constructor was written by Mr T. Bealor.
4.2.2 Query Types Studied

We chose 12 types of queries to test our algorithm as shown in Table 4.2. For each type of query, the size of the domain was always between 500 and 600 and we tested 50 random queries as generated by the query generator described in [Bea95].

<table>
<thead>
<tr>
<th>Query type</th>
<th>Number of relations</th>
<th>Number of attributes</th>
<th>Connectivity (%)</th>
<th>Size of relation</th>
<th>Selectivity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>1 - 3</td>
<td>60 - 70</td>
<td>1500-2000</td>
<td>0.9 - 1.0</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1 - 3</td>
<td>60 - 70</td>
<td>1500-2000</td>
<td>0.6 - 1.0</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>1 - 3</td>
<td>60 - 70</td>
<td>5000-6000</td>
<td>0.9 - 1.0</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>1 - 3</td>
<td>60 - 70</td>
<td>5000-6000</td>
<td>0.6 - 1.0</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>1 - 4</td>
<td>40 - 50</td>
<td>1500-2000</td>
<td>0.9 - 1.0</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>1 - 4</td>
<td>40 - 50</td>
<td>1500-2000</td>
<td>0.6 - 1.0</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>1 - 4</td>
<td>40 - 50</td>
<td>5000-6000</td>
<td>0.9 - 1.0</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
<td>1 - 4</td>
<td>40 - 50</td>
<td>5000-6000</td>
<td>0.6 - 1.0</td>
</tr>
<tr>
<td>9</td>
<td>6</td>
<td>1 - 4</td>
<td>40 - 50</td>
<td>1500-2000</td>
<td>0.9 - 1.0</td>
</tr>
<tr>
<td>10</td>
<td>6</td>
<td>1 - 4</td>
<td>40 - 50</td>
<td>1500-2000</td>
<td>0.6 - 1.0</td>
</tr>
<tr>
<td>11</td>
<td>6</td>
<td>1 - 4</td>
<td>40 - 50</td>
<td>5000-6000</td>
<td>0.9 - 1.0</td>
</tr>
<tr>
<td>12</td>
<td>6</td>
<td>1 - 4</td>
<td>40 - 50</td>
<td>5000-6000</td>
<td>0.6 - 1.0</td>
</tr>
</tbody>
</table>

Table 4.2 Query types

4.3 Experimental Results

The results of our experiments are shown in Table 4.3. We have used three strategies for processing a query — no optimization, AHY and MJ. For the AHY and the MJ strategy, we have estimated the cost of query processing in two
EXPERIMENTS AND EVALUATION

ways — using statistical techniques and by simulating operations on a synthetic database. Thus we have a total of 5 techniques for getting the cost of processing a query. An entry in any of the columns 2 — 6 gives the average cost to process 50 queries of a given query type using a specified technique as described below:

<table>
<thead>
<tr>
<th>Column 1: Query types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Column 2: No query optimization is done. We calculate the cost of processing a query by computing the sum of the costs to send each participating relation to the query site.</td>
</tr>
<tr>
<td>Column 3: We estimate the cost, based on statistical information, for processing a query using the AHY Algorithm (total time version).</td>
</tr>
<tr>
<td>Column 4: We simulate the schedule generated by the AHY Algorithm by carrying out the corresponding database operations on a synthetic database to get an accurate estimate of the actual cost.</td>
</tr>
<tr>
<td>Column 5: We estimate the cost, based on statistical information, for processing the query using the MJ Algorithm.</td>
</tr>
<tr>
<td>Column 6: We simulate the schedule generated by the MJ Algorithm by carrying out the corresponding database operations on a synthetic database to get an accurate estimate of the actual cost.</td>
</tr>
</tbody>
</table>
EXPERIMENTS AND EVALUATION

<table>
<thead>
<tr>
<th>Type</th>
<th>No optimization</th>
<th>AHY (estimated cost)*</th>
<th>AHY (actual cost)*</th>
<th>MJ (estimated cost)</th>
<th>MJ (actual cost)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>15615</td>
<td>16226</td>
<td>15615</td>
<td>4600</td>
<td>4892</td>
</tr>
<tr>
<td>2</td>
<td>15646</td>
<td>12458</td>
<td>12351</td>
<td>3612</td>
<td>3791</td>
</tr>
<tr>
<td>3</td>
<td>49588</td>
<td>47657</td>
<td>47574</td>
<td>13580</td>
<td>17567</td>
</tr>
<tr>
<td>4</td>
<td>49267</td>
<td>33795</td>
<td>33696</td>
<td>9488</td>
<td>13121</td>
</tr>
<tr>
<td>5</td>
<td>30024</td>
<td>26474</td>
<td>31001</td>
<td>5897</td>
<td>5782</td>
</tr>
<tr>
<td>6</td>
<td>30356</td>
<td>16774</td>
<td>20078</td>
<td>4142</td>
<td>4089</td>
</tr>
<tr>
<td>7</td>
<td>95186</td>
<td>88067</td>
<td>88105</td>
<td>22516</td>
<td>33720</td>
</tr>
<tr>
<td>8</td>
<td>94423</td>
<td>47412</td>
<td>46790</td>
<td>17306</td>
<td>22107</td>
</tr>
<tr>
<td>9</td>
<td>35555</td>
<td>30432</td>
<td>35690</td>
<td>3874</td>
<td>4099</td>
</tr>
<tr>
<td>10</td>
<td>35831</td>
<td>15902</td>
<td>19656</td>
<td>2873</td>
<td>3116</td>
</tr>
<tr>
<td>11</td>
<td>111719</td>
<td>96374</td>
<td>94771</td>
<td>16222</td>
<td>21452</td>
</tr>
<tr>
<td>12</td>
<td>109348</td>
<td>37732</td>
<td>36570</td>
<td>8328</td>
<td>10384</td>
</tr>
</tbody>
</table>

* These costs were calculated using the software written by Mr Todd Bealor [Bea95]. In situations where the estimated cost using AHY is higher than the cost without any optimization, we don’t use any optimization.

Table 4.3 Experimental results

We now compare, for each of the twelve query categories, the improvements using the AHY strategy to the improvements using the MJ strategy. We calculate the improvements as follows:

Improvement using the AHY Algorithm = \left(1 - \frac{\text{AHY actual cost}}{\text{Cost without any optimization}}\right) \times 100\%

Improvement using the MJ Algorithm = \left(1 - \frac{\text{MJ actual cost}}{\text{Cost without any optimization}}\right) \times 100\%

Improvement of MJ over AHY = \text{Improvement using MJ} - \text{Improvement using AHY}
EXPERIMENTS AND EVALUATION

<table>
<thead>
<tr>
<th>Type</th>
<th>Improvement of AHY Algorithm (%)</th>
<th>Improvement of MJ Algorithm (%)</th>
<th>Improvement of MJ over AHY (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>none</td>
<td>68.7</td>
<td>68.7</td>
</tr>
<tr>
<td>2</td>
<td>21.1</td>
<td>75.8</td>
<td>54.7</td>
</tr>
<tr>
<td>3</td>
<td>4.1</td>
<td>64.6</td>
<td>60.5</td>
</tr>
<tr>
<td>4</td>
<td>31.6</td>
<td>73.4</td>
<td>41.8</td>
</tr>
<tr>
<td>5</td>
<td>none</td>
<td>80.7</td>
<td>80.7</td>
</tr>
<tr>
<td>6</td>
<td>33.9</td>
<td>86.5</td>
<td>52.6</td>
</tr>
<tr>
<td>7</td>
<td>7.4</td>
<td>64.6</td>
<td>57.2</td>
</tr>
<tr>
<td>8</td>
<td>50.4</td>
<td>76.6</td>
<td>26.2</td>
</tr>
<tr>
<td>9</td>
<td>none</td>
<td>88.5</td>
<td>88.5</td>
</tr>
<tr>
<td>10</td>
<td>45.1</td>
<td>91.3</td>
<td>46.2</td>
</tr>
<tr>
<td>11</td>
<td>15.2</td>
<td>80.8</td>
<td>65.6</td>
</tr>
<tr>
<td>12</td>
<td>66.6</td>
<td>90.5</td>
<td>23.9</td>
</tr>
</tbody>
</table>

Table 4.4 Improvement of AHY and MJ algorithms

4.4 Critical Analysis of The Results

The experimental results in the last section indicate that the MJ Algorithm, which combines MASJ operation with AHY Algorithm, has substantially improved the cost of processing a wide range of SPJ type queries. Our analysis of the results is given below:

- When the selectivity was varied between 0.9 and 1.0, the AHY Algorithm had none or very little improvement, but the MJ Algorithm still had over 60%
improvement (Figure 4.1).

![AHY vs. MJ](image)

Figure 4.1 Queries with selectivity 0.9 — 1.0

- When the selectivity was varied between 0.6 and 1.0, the MJ Algorithm still had significant improvement over the AHY Algorithm (Figure 4.2).
Using the MJ Algorithm, if we fixed other parameters and varied connectivity between 40% — 50% and 60% — 70%, we found that the queries with a higher connectivity got a better improvement (Figure 4.3).
The ratio of the number of tuples in a relation and the domains of those attributes of the relation were also somewhat important for the performance of MJ Algorithm. The smaller the ratio, the better the improvement (Figure 4.4).
Queries with different sizes of relations

Figure 4.4 Queries with different number of tuples in relations
Chapter 5 CONCLUSIONS

In this thesis, we have introduced the notion of a multi-attribute semijoin (MASJ) operation and have studied the properties of this operation. We have carried out experiments to determine whether this operation is useful for query optimization in distributed databases. The multi-attribute semijoin operation shares the characteristic of join operations that, after the operation, result in a relation containing new attributes. This MASJ operation, like semijoin operations, does not involve the transmission of non-join attributes. In general, this is a multi operand operation with $n$ operands, where $n \geq 2$. In this operation, more than one attribute is sent and the size of a relation is reduced by eliminating the combinations of values of attributes.

To study the usefulness of this operation in distributed query processing, we have modified a well known heuristic for query processing, the AHY algorithm, to include this operation. We call the modified algorithm the MJ Algorithm. This algorithm is a static heuristic which combines the multi-attribute semijoin operation with the AHY Algorithm (total time version). To study the relative performances of schedules of database operations generated by the MJ Algorithm and those produced by the AHY Algorithm, we designed experiments to test
different types of queries. In total we studied 12 categories of queries, each with a different characteristic. For each query category, we generated a large number of queries using [Bea95]. We tested each query by generating two schedules — one using the MJ algorithm and the other using the AHY Algorithm. We then simulated each of these schedules with a synthetic database containing data generated randomly. Our experimental results indicate that the MJ Algorithm outperforms the AHY Algorithm quite significantly.

We would like to point out that the MJ algorithm is not intended to be a definitive heuristic for query processing. There is no single best strategy for query optimization and, depending on the database profile and the query graph, there are situations where each of the strategies using join, semijoin, multi-attribute semijoin or bloom join may be the best strategy. We conclude that optimization heuristics can be improved by taking into account all the characteristics of both the query as well the database profile and applying the most suitable technique.
## APPENDIX A

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>R</td>
</tr>
<tr>
<td>$</td>
<td>R(A)</td>
</tr>
<tr>
<td>$</td>
<td>A</td>
</tr>
<tr>
<td>$\omega_T$</td>
<td>Sum (in bytes) of size of all attributes in the set of attributes $T$</td>
</tr>
<tr>
<td>$d_{ij}$</td>
<td>Attribute $j$ in relation $i$</td>
</tr>
<tr>
<td>$R(T)$</td>
<td>The projection of relation $R$ on a set of attributes $T$</td>
</tr>
<tr>
<td>$R_i \Rightarrow R_j$</td>
<td>A natural join of relations $R_i$ and $R_j$</td>
</tr>
<tr>
<td>$R_j \leftarrow A \rightarrow R_i$</td>
<td>The semijoin from relation $R_j$ to relation $R_i$ on attribute $A$</td>
</tr>
<tr>
<td>$G = (V, E)$</td>
<td>Join query graph $G$, where $V$ is the set of nodes and $E$ is the set of edges in $G$</td>
</tr>
<tr>
<td>$L$</td>
<td>The set of labels of all edges of $E$ in graph $G$</td>
</tr>
<tr>
<td>$\text{parent}(N)$</td>
<td>In a tree, $\text{parent}(N)$ is the node immediately preceding $N$ in the path from the root to $N$</td>
</tr>
<tr>
<td>$\otimes (j, S)$</td>
<td>Multi-attribute semijoin operation for subquery graph $S$ and will be carried out at site $j$</td>
</tr>
<tr>
<td>$\text{ATTR}(S)$</td>
<td>The set of all edge labels in subquery graph $S$</td>
</tr>
<tr>
<td>$\text{SCHEMA}(R_i)$</td>
<td>The set of all attributes in $R_i$</td>
</tr>
<tr>
<td>$M_{\text{tuple}}$</td>
<td>Number of tuples resulting from a MASJ operation</td>
</tr>
<tr>
<td>$n \neq i$</td>
<td>$n$ is not equal to $i$</td>
</tr>
</tbody>
</table>
APPENDIX B

Given a query graph $G = (V, E)$. $S = (V_s, E_s)$ is subquery graph of $G$. Suppose $\otimes (j, S)$ will be carried on $S$ in node $j$, node $i$ is reduced by $R_x(A_x)$, $x \in V - V_s$, $R_1, R_2, \ldots, R_p$ are the relations corresponding to nodes in $V_s$, $A_1, A_2, \ldots, A_q$ are the distinct attributes associated with edges in $E_s$, $m_i$ is the number of different nodes that edges with attribute $A_i$ are incident to. The resulting relation ($M_{tuple}$) of joining $R_1, R_2, \ldots, R_p$ can be calculated as following:

$$M_{tuple} = \frac{\prod_{i=1}^{q} |A_i|^{(m_i - 1)} \times |A|}{\prod_{i=1}^{p} |R_i| \times |R_x(A)|}$$  \hspace{1cm} (B.1)
APPENDIX C

Heuristic H: Given a query graph $G = (V, E)$, find each cycle which has edges with different labels.

Procedure $H(G)$
For all $\alpha \in V$ do
  currentLevel $\leftarrow \alpha$
  setOfNodesInTree $\leftarrow \alpha$
  while currentLevel $\neq$ NULL do
    newLevel $\leftarrow$ NULL
    for all $\alpha \in$ currentLevel do
      for all UNMARKEDEDGE(v, w) $\in$ E do
        if label(v, w) $\neq$ label(parent(v), v) then
          call MARKEDGE(v, w)
          if w $\in$ setOfNodesInTree then
            if $\alpha =$ FINDFIRSTCOMMONANCESTOR(v, w) then
              call COPYCYCLE(v, w)
              end
            else
              call INSERTASCHILD(v, w)
            end
            newLevel $\leftarrow$ newLevel $\cup$ w
            setOfNodesInTree $\leftarrow$ setOfNodesInTree $\cup$ w
          end
        end
      end
    currentLevel $\leftarrow$ newLevel
  end
  for all EDGE($\alpha$, w) $\in$ E do
    E $\leftarrow$ E $-$ (v, w)
  end
end H
APPENDIX D

**RSC** — the cost of sending data from relation $R_j$ reduced by SCH, it is calculated as follows:

$$RSC = \text{the tuples of relation reduced by SCH}$$  \hspace{1cm} (D.1)

$$\times \omega_1^{\text{SCHEMA}(R_j) \cap \text{ATTR}(Si)}$$

**SC** — the cost of sending data from original relation $R_j$, it is calculated as following:

$$SC = \text{the tuples of original relation}$$  \hspace{1cm} (D.2)

$$\times \omega_1^{\text{SCHEMA}(R_j) \cap \text{ATTR}(Si)}$$

Determine the cost of sending data from a relation for MASJ operation by the following procedure:

For each relation $R_j$ in $Si$. do

If(cost of SCH + RSC < SC)

transfering cost = RSC  \hspace{1cm} (D.3)

ELSE

transfering cost = SC
APPENDIX E

Details of Example 3.4

We now show how the steps given in example 3.4 are actually carried out.

Step 1: Find all candidate cycles

To simplify the description, we will show how to find one cycle using heuristic H. To do this we will only show how one applicable edge from each node is processed.

- The process starts from node 0 which has three edges — 2 with labels A to nodes 1 and 3 and one with a label D to node 1. The process goes the edge labelled A to node 3. We have a path (0 — A — 1)

- Node 1 has three edges — two edges with a label B to nodes 2 and 4, one with a label D to node 0 . The process goes the edge labelled D to node 0. We now get the first cycle (0 — A — 1 — D — 0) which is S3 in Figure 3.

- The process keeps this cycle and continues its trip with other branches. After finishing searching from node 0, it will start from node 1, then 2 and 3. When going through all searching, it finds three cycles as below:

\[ S1 = \{R_0, R_1, R_2, R_3\}, E1 \]  \[ L1 = \{A, B, C, D\} \]

\[ S2 = \{R_1, R_2, R_3\}, E2 \]  \[ L2 = \{A, B, C\} \]

\[ S3 = \{R_0, R_1\}, E3 \]  \[ L3 = \{A, D\} \]
and their subquery graphs are showed in Figure E.1.

Step 2: Determine BC, the site of the MASJ operation and generate the MSCH schedule

The AHY Algorithm is applied to query G to generate a schedule for each relation as shown in Figure E.2. If they are used as reducers to reduce all relations in G,
The size of all relations after reduction are

$$|R_0| = 1918. \quad |R_1| = 2160. \quad |R_2| = 3921. \quad |R_3| = 1764. \quad |R_4| = 4624$$

Now calculate, for each of the cycles, the cost and benefit for the schedule using our MASJ operations. We show the details only for cycle $S1$. 

---


Figure E.2 Schedules for relations in $G$
cycle $S_1$:

Cost of transferring data from

$R_0 : \quad RSC = 1918 \times 2 = 3836$

$R_1 : \quad RSC = 2160 \times 2 = 4320$

$R_2 : \quad RSC = 3921 \times 2 = 7842$

$R_3 : \quad RSC = 1764 \times 2 = 3528$

since the cost of $SCH + RSC < SC$ from each relation in $S_1$. The site of the MASJ operation for this cycle should be site 2 since transferring data from $R_2$ is the maximum.

the net benefit of $MSCH = \sum_{i=0}^{3} (\text{the benefit of reducing } R_i - \text{the cost of reducing } R_i)$

$= -29342$

$F = \frac{\text{the net benefit of } MASH}{4} < 0$

This cycle will be deleted since $F < 0$.

cycle $S_2$:

The site of MASJ operation should be site 2 since transferring data from $R_2$ (7842) cost the most (from $R_1$ (4320) and from $R_3$ (3528) in $S_2$.

the net benefit of $MSCH = 14009$

$F = 4669$
This cycle is kept as a candidate cycle which MASJ operation may be carried on.

**cycle S3:**

The site of MASJ operation should be site 1 since transferring data from $R_I$ (4320) costs more than from $R_0$ (3836) in S3.  

the net benefit of $\text{MCSH} = 12530$

$$F = 6265$$

This cycle is chosen as best cycle to carry out MASJ operation since $F$ of S3 is greater than $F$ of S2. The site of operation should be site 1.

The MSCH is showed in Figure E.3.

1) A reducer that is generated by the AHY Algorithm is used to reduce $R_0$.

2) Reduced $R_0(A, D)$ is sent to site 1 to reduce $R_I$.

3) Reduced $R_I(A, D)$ is sent back to site 0 to reduce $R_0$ again

This is a serial procedure, and the order of sending data is important.

Finally, the MSCH is appended to $\text{QSCP}$. 

![Diagram](image)

Figure E.3 Schedules for $R_0$ and $R_I$
Step 3: Generate new reducers to reduce the size of relations not appearing in BC

| $R_i$ | $|R_i|_i$ | X     | $|R_i(X)|_i$ |
|-------|----------|-------|-------------|
| $R_0$ | 80       | A     | 80          |
|       |          | D     | 80          |
|       |          | E     | 80          |
| $R_1$ | 78       | A     | 78          |
|       |          | B     | 78          |
|       |          | D     | 78          |
|       |          | F     | 78          |
| $R_2$ | 5990     | B     | 505         |
|       |          | C     | 465         |
|       |          | G     | 500         |
| $R_3$ | 5140     | A     | 428         |
|       |          | C     | 425         |
|       |          | H     | 500         |
| $R_4$ | 5780     | B     | 555         |
|       |          | I     | 500         |

Table E.1 Database statistics

Now based on the new database statistics (Table E.1), update candidate schedules using step 2 of the AHY Algorithm and integrate those candidate schedules for $R_2$, $R_3$, and $R_4$ using step 3 of the AHY Algorithm. These new schedules for $R_2$, $R_3$, and $R_4$ are shown in Figure E.4.
Obviously, these reducers are better than those reducers shown in Figure E.2 for $R_2$, $R_3$, and $R_4$.

**Step 4: Check the remaining cycles in $G$**

In the example, $S_2$ is the only remaining cycle, but $V_2 \cap V_3! = \emptyset$, it means $S_2$ includes some nodes which are also included in $S_3$ that is BC, so the calculation terminates after appending schedules in Figure E.4 to $QSCH$. 
BIBLIOGRAPHY


VITA AUCTORIS

Qiuling Fu was born in 1962 in Beijing, China. She graduated from High School in 1980. From there she went on to the Beijing Institute of Technology, China where she obtained a B. Sc. in Engineering in 1984. She is currently a candidate for the Master’s degree in Computer Science at the University of Windsor and will graduate in June of 1996.